Autonomous Robot Dancing Driven by Beats and Emotions of Music

Guangyu Xia
Computer Science Department
Carnegie Mellon University
Pittsburgh, PA 15213, USA
gxia@andrew.cmu.edu

Junyun Tay*
Mechanical Engineering Department
Carnegie Mellon University
Pittsburgh, PA 15213, USA
junyunt@cmu.edu

Roger Dannenberg
Computer Science Department
Carnegie Mellon University
Pittsburgh, PA 15213, USA
rbd@cs.cmu.edu

Manuela Veloso
Computer Science Department
Carnegie Mellon University
Pittsburgh, PA 15213, USA
veloso@cmu.edu

ABSTRACT

Many robot dances are preprogrammed by choreographers for a particular piece of music so that the motions can be smoothly executed and synchronized to the music. We are interested in automating the task of robot dance choreography to allow robots to dance without detailed human planning. Robot dance movements are synchronized to the beats and reflect the emotion of any music. Our work is made up of two parts: (1) The first algorithm plans a sequence of dance movements that is driven by the beats and the emotions detected through the preprocessing of selected dance music. (2) We also contribute a real-time synchronizing algorithm to minimize the error between the execution of the motions and the plan. Our work builds on previous research to extract beats and emotions from music audio. We created a library of parameterized motion primitives, whereby each motion primitive is composed of a set of keyframes and durations and generate the sequence of dance movements from this library. We demonstrate the feasibility of our algorithms on the NAO humanoid robot to show that the robot is capable of using the mappings defined to autonomously dance to any music. Although we present our work using a humanoid robot, our algorithm is applicable to other robots.

Categories and Subject Descriptors

I.2.8 [**Problem Solving, Control Methods, and Search**]: Plan execution, formation, and generation

General Terms

Algorithms

Keywords

autonomous robot dancing, motion-emotion mapping, scheduling, real-time synchronization

*Junyun Tay is in the Carnegie Mellon University-Nanyang Technological University Dual PhD Programme.

Appears in: *Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2012)*, Conitzer, Winikoff, Padgham, and van der Hoek (eds.), 4-8 June 2012, Valencia, Spain.

1. INTRODUCTION

Dancing motions for robots are usually created by choreographers and designed for a particular piece of music. If the piece of music changes, the dance movements of the robot will have to be recreated. We are interested in automating the task of robot dance choreography by generating sequences of dance movements from a motion library. The automatically generated choreography should satisfy several goals. First, the choreography should be safe for performance. For example, it should not cause the robot to fall or break. Second, the choreography should reflect the emotional character of the music. Peaceful music should be choreographed differently from music that sounds angry. Third, the dance should be synchronized to the music. Finally the dance should not be deterministic. Even when the emotion and tempo of the music remain constant, the dance should contain interesting variations.

To address the goal of safety, we compose dances from sequences of motion primitives. A motion primitive is a sequence of keyframes (static poses), interpolated to form a continuous motion. The motion primitives are designed to be interesting and safe when performed in any sequence.

We represent emotion using a two-dimensional activation-valence emotion space, which is commonly used to describe emotional states. We create a large library of motion primitives, by dividing the joints of the NAO humanoid robot into 4 categories, where each category of joints can actuate independently. Given that there is a large number of motion primitives, we contribute another algorithm that uses labelled emotional static postures data, collected with the NAO humanoid robot, to estimate the activation-valence values for each motion primitive. Motion primitives are then selected to match the emotional state of the music.

To synchronize dance to the music, we use the fact that motion primitives can be executed with different durations. We adjust the duration of each motion primitive so that the duration will be an integer multiple of beats. In practice, as the movements on the NAO humanoid robot may not execute according to the planned schedule of motion primitives, we maintain synchronization with the music by adjusting the durations of motion primitives in real time to compensate for differences between the schedule and the actual execution.

To create interesting variations in the dance, we use a first-order Markov model to generate dances stochastically. States correspond to motion primitives. The state transition probabilities are designed to produce smooth motion sequences by favoring next states that begin with a keyframe near the final keyframe of the current state. The state transition probabilities also depend upon the current emotion in the music, such that at any given time, state transition probabilities will prefer states that reflect the current emotion in the music. Figure 1 summarizes the process we described. We demonstrate the feasibility of our algorithms on the NAO humanoid robot to show that the robot is capable of using the mappings defined to autonomously dance to any music. Our algorithms are applicable to other robots as well.

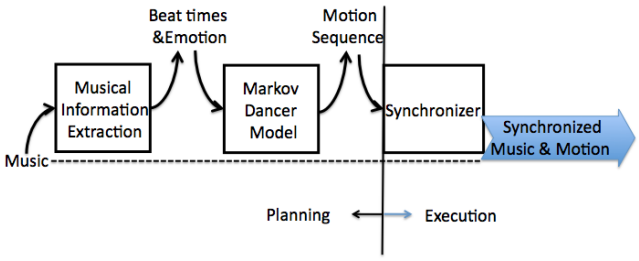


Figure 1: Summary of system diagram

2. RELATED WORK

Robot dances are generally preprogrammed by choreographers specifically for a piece of music. Researchers created motions for robots using motion capture data of humans dancing [13]. Our algorithm enables the robot to dance autonomously to any piece of music that is preprocessed using a library of predefined motions. Many explored emotional expressions of robots using the robot's facial features [2, 10] and did not consider using entire body postures and movements. Paul Ekman proposed 6 primary emotions: Happy, Sad, Angry, Surprised, Fear and Disgust [3], and we explore the use of body postures on humanoid robots to express 6 primary emotions. We use the information collected from these static emotion body postures to estimate the emotion of dance movements and organized them to reflect the emotions detected in music.

The creation of dancing characters is a common theme in computer animation research [8, 9, 11, 14, 15]. [11] uses a given sequence of motions to synthesize music, while others use given music to synthesize motion sequences. The work [14] focuses on a topological model of dance styles, while [8, 9, 15] focus on synthesizing motions according to musical beat times, which are very similar to our approach. Among them, only [15] synthesizes motions according to musical emotion. However their system uses only the intensity of music as an indicator of emotion. Our work enhances the use of beats by adopting a more recent and more accurate technique to identify beats and tempo. We also use a state-of-the-art emotion detection system coupled with our motion primitives for robots to convey richer emotions.

3. DANCE MOTIONS FORMALIZATION

We demonstrate our work on a NAO humanoid robot (Figure 2). The NAO robot has 21 joints and is a stand-alone autonomous robot with no facial features, except for LEDs in the eyes and ears. We group the joints into 4 categories:

1. Head (*Head*): HeadYaw, HeadPitch
2. Left Arm (*LArm*): LShoulderPitch, LShoulderRoll, LElbowYaw, LElbowRoll
3. Right Arm (*RArm*): RShoulderPitch, RShoulderRoll, RElbowYaw, RElbowRoll
4. Legs (*Legs*): LHipYawPitch, LHipRoll, LHipPitch, LKneePitch, LAnklePitch, LAnkleRoll, RHipRoll, RHipPitch, RKneePitch, RAnklePitch, RAnkleRoll

Figure 2: NAO humanoid robot

Each category of joints is defined to be $c = \langle J_{c,1}, \ldots, J_{c,|c|} \rangle$, where $c \in \{Head, LArm, RArm, Legs\}$ and $J_{c,1}, \ldots, J_{c,|c|}$ are the indices of the joints in the category. $|c|$ is the total number of joints in the category c. E.g., $Head = \langle 1, 2 \rangle$ where 1 is the index of HeadYaw and 2 is the index of HeadPitch.

3.1 Motion Primitive

Each keyframe is associated with a category c, and is defined as $K_c = \langle V_{c,1}, \ldots, V_{c,|c|} \rangle$, where $V_{c,j}$ contains the joint angle of joint index $J_{c,j}$. A motion primitive is $M_c(\beta) = \langle K_{c,1}, \beta D_1, K_{c,2}, \ldots, K_{c,F-1}, \beta D_{F-1}, K_{c,F} \rangle$ where F is the number of keyframes in M_c. D is the minimum time that it takes to move (interpolate) from one keyframe, $K_{c,f}$, to the next keyframe, $K_{c,f+1}$, and is pre-defined. We parameterize the motion primitive with β, where $\beta \in \mathbb{R}$ and $\beta \geq 1$. β is calculated using the beat times of the music so as to synchronize the motion primitive with the music (Section 5.2).

We assume independence of each body part by ignoring the effects of dynamics generated by motions. This categorization of joints according to body parts enables us to create a large variety of motion primitives, as we can create motion primitives for each category independently. We have a total of $8(Head) \times 9(LArm) \times 9(RArm) \times 26(Legs) = 16,848$ possible motion primitive combinations. The number of motion variations is actually much greater because motions primitives do not necessarily start and end synchronously. The 52 parameterized motion primitives are manually generated.

3.2 Schedule of Motion Primitives

A schedule of motion primitives belonging to the category c is defined as $S_c = \langle K_c, \mathcal{D}, M_{c,1}, I_1, M_{c,2}, \ldots, I_{p-1}, M_{c,p} \rangle$, where p is the total number of motion primitives in S_c. K_c contains the initial joint angles of the robot and $\mathcal{D}$ is the time to interpolate from K_c to the first keyframe of $M_{c,1}$. I_m is the time to interpolate from the last keyframe of $M_{c,m}$ to the first keyframe of $M_{c,m+1}$. We show how to calculate I_m in Section 5.2.1. We plan 4 schedules of motion primitives—$S_{Head}, S_{LArm}, S_{RArm}, S_{Legs}$ independently according to the emotions and beats of the music. Although the 4 schedules are generated independently, the robot can execute these 4 schedules simultaneously. The behaviour of the robot at a given point in time h is $B_h = \langle M_{Head}, M_{LArm}, M_{RArm}, M_{Legs} \rangle$

where M_c is the current motion primitive in S_c at time h where $c \in \{Head, LArm, RArm, Legs\}$.

Our formalization of the motion primitive, schedules of motion primitives and behaviour is general to use on different robots given the independence of the joints of the robot and that the joints can be actuated simultaneously.

4. MUSIC INFORMATION EXTRACTION

We obtain information directly from music audio signals to coordinate the robot dance motions with music. The extracted information consists of *emotions*, so that the robot's motions can be consistent with the mood of the music, and *beats*, which allow the robot to synchronize to musical pulses.

4.1 Emotion Extraction

It is well known that musical emotion has a significant impact on human dancers' movements. Our autonomous robot dance algorithm is driven by musical emotions.

We use SMERS [7], a state-of-the-art music emotion recognition system based on audio features and support vector regression (SVR). SMERS performs a forced classification into 11 emotion categories and achieved a 94% agreement with expert human labelers.

4.1.1 Emotion Representation

SMERS adopts Thayer's 2-dimensional Activation-Valence (AV) model [16] to represent musical emotion (Figure 3).

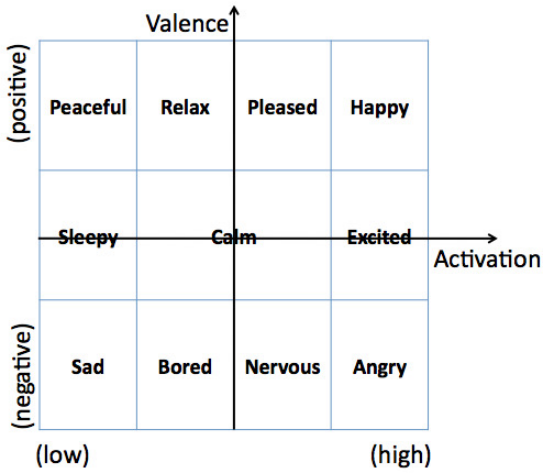


Figure 3: 2-dimensional emotion model

The emotion of a piece of music is represented by an AV value, (a, v) where a denotes Activation and v denotes Valence; $a, v \in \mathbb{R}, a \in [-1, 1], v \in [-1, 1]$. SMERS was developed and tested assuming each track of music (song) has a single main emotion. We consider that emotion may change over time within a piece, so we use a 30-second sliding window with a 15-second overlap. Therefore, music emotion is represented by a vector of (a, v) coordinates. The i^{th} element in the vector (indexing starts at 0) represents the emotion of the music at time $15i + 15$ seconds.

4.1.2 SVR training and decoding

Emotion labeling relies on SVR, which learns a mapping between feature vectors and emotion vectors. In our application, the feature vector $x_i \in \mathbb{R}^6$ is a vector of extracted music audio features, which include estimated key (one of 12 major or minor keys), average energy and standard deviation of energy, estimated tempo, standard deviation of beat duration, and harmonicity (see [7] for more details). The emotion vector is an (a, v) coordinate denoted by $y_i \in \mathbb{R}^2$. Since the original training data contains music audio with emotion labels such as Peaceful or Happy, we replace these labels with the (a, v) coordinates of the middle of the corresponding block as shown in Figure 3. Given a set of training data $(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)$, SVR will try to find the optimal mapping function between input x_i and output y_i. In the decoding step, for a segment of music audio, we extract the feature vector x and apply the learned regression model to get the output (a, v) coordinates of the piece of music. These coordinates could be quantized to obtain a discrete label (e.g. Angry), but for choreography, we use the continuous numerical representation directly.

4.2 Beat Tracking

Musical beats reflect the basic period or pulse of music. It is typical for humans to synchronize the dance motions to the musical beats, so it is important for our algorithm to detect and track beats in music audio.

Much work has been done in the area of beat tracking. Currently, most algorithms are based on autocorrelation analysis or onset component detecting [1, 4, 5, 6]. Most recently, [1] proposed a beat tracking method that combines autocorrelation analysis and Neural Networks learning. The work done by Goto [6] is based on onset component and rhythm structure analysis. In nearly all cases, beat detection is based on some audio feature associated with note onsets and drum beats, such as change in amplitude or spectrum. Peaks in these features mark likely candidates for beat locations. Since musical beats mostly occur with an overall stable frequency (tempo), these candidate locations are filtered by looking for ones that are regularly spaced. Our approach estimates a global tempo (the overall beats per minute of a piece of music) by analyzing the autocorrelation of onset features throughout a piece, and then finds the best beat times by using dynamic programming [4]. This method performed well in the MIREX-06 evaluations [4] and generally works well when there are clear beats and steady tempo.

4.2.1 Global tempo estimation

The global tempo estimation algorithm is executed in three steps. First, the onset strength envelope (Figure 4) of the whole piece of music is calculated from a crude perceptual model (see [4] for more details). Second, the autocorrelation of the onset strength is computed. When the lag matches the beat or its multiples, the autocorrelation should be closer to 1. The result of one piece of music is shown in Figure 5. Third, the highest peak (ignoring the peak at 0) is detected and the corresponding lag is chosen as the global tempo.

4.2.2 Find best beat times

Local beat times correspond to perceived onsets in the audio signal of a piece of music. Given the global tempo, the beat times should not only be local onset peaks but should be equally spaced according to the global tempo. An objective function is defined to reflect these two goals.

$$S(\boldsymbol{T}) = \alpha \sum_{i}^{N} Onset(t_i) - (1 - \alpha) \sum_{i+1}^{N} dist(t_i - t_{i-1}, C) \quad (1)$$

Here, $\boldsymbol{T} = [t_1, t_2, \ldots, t_N]$ refers to the sequence of beat

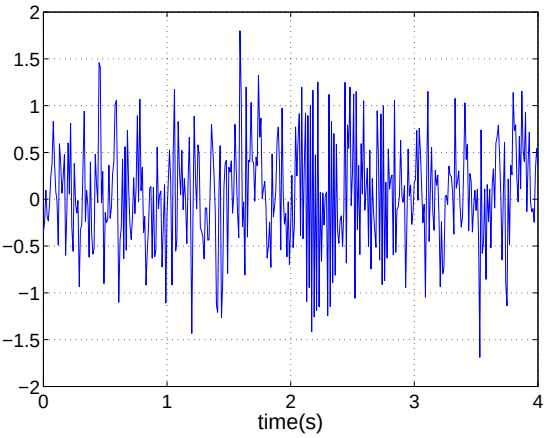


Figure 4: Part of the onset envelope of the music

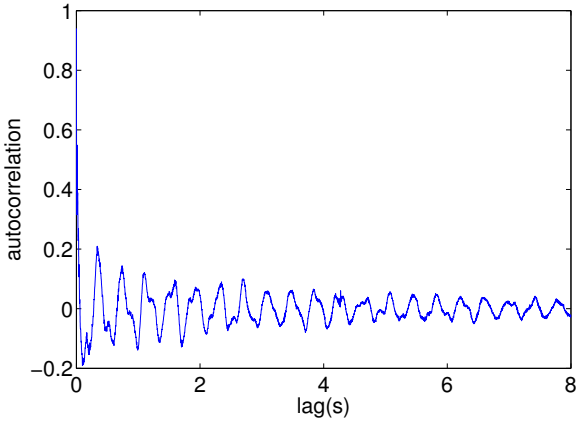


Figure 5: Autocorrelation as function of lag

times. $\alpha \in (0, 1)$ is a weighing parameter to balance the importance of the two terms. $Onset(t_i)$ represents the onset strength of the audio signal at time t_i, while $dist(t_i - t_{i-1}, C)$ represents the difference between the beat interval and the global tempo C, which is determined as described above. We use dynamic programming to optimize the objective function by recursively finding each t_i and hence the best $\boldsymbol{T}$.

5. DANCING PLANNING

In this section, we explain how musical emotions and beat times automatically determines the behaviour of the robot. The musical emotion decides the motion primitive, while beat times control fine timing and synchronization.

5.1 Generate Schedule of Motion Primitives

To generate a schedule of motion primitives according to the emotions of the music, we need to have emotion labels assigned to the motion primitives. It is time-consuming to label each motion primitive with an AV value, so we developed an algorithm that estimates the AV value from emotion labels of the static postures within the motion primitive.

5.1.1 *Mapping Motion Primitive to Activation-Valence Space From Static Postures Data*

We collected 4 static postures of the NAO humanoid robot for each of Ekman's 6 basic emotions: Happy, Sad, Angry, Surprised, Fear and Disgust. Hence, we have a total of 24 emotional static postures. Motion primitives are contructed from these. The arms and head of a NAO humanoid robot can be freely positioned, but there are only 5 different heights and 5 different tilts of the robot to choose from as shown in Figure 6. Static postures to express each emotion using the NAO humanoid robot were collected independently. Figure 7 shows a subset of the data we collected.

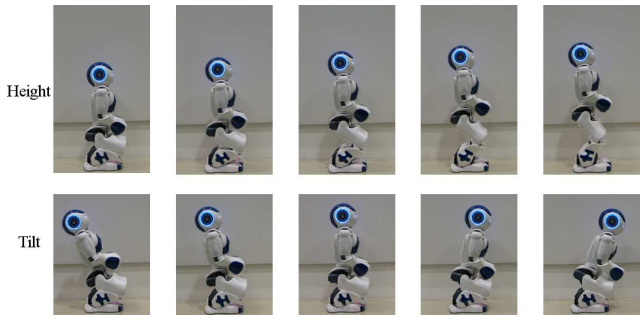


Figure 6: 5 heights and 5 tilts of the NAO robot

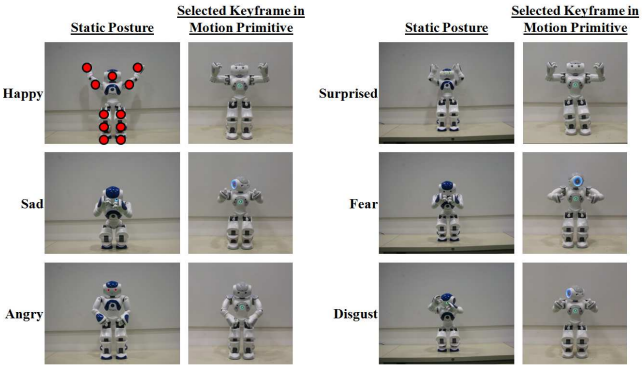


Figure 7: Examples of static postures and corresponding selected keyframes in motion primitives. Red circles indicate the points of interest

Table 1: AV Value for Paul Ekman's six emotions

Emotion	Activation (A)	Valence(V)
Happy	1	1
Sad	-1	-1
Angry	1	-1
Surprised	1	0
Fear	0.5	-1
Disgust	-0.5	-0.5

We assigned each of the 6 basic emotions with an AV value shown in Table 1. We recorded the joint angles of each emotional static posture in a keyframe. To label each motion primitive, we form a weighted sum based on the similarity between motion primitives and static postures with known AV values. SP[em] returns a vector of four static postures (keyframes), where em $\in$ {Happy, Sad, Angry, Disgusted, Fear, Surprised}, e.g., SP[Happy] returns a vector of four keyframes that reflect the Happy emotion. Algorithm 1 first determines the most similar static posture for each *em* to the motion primitive by using the points of interest (POI),

shown in Figure 7. The POI are chosen based on the concepts of markers in motion capture systems. Algorithm 1 calculates DIST, the sum of the Euclidean distances between the 3-dimensional positions of the POI in each static posture with emotion *em* and the 3D positions of POI of the keyframes in the motion primitive $M_{c,n}$ with the function GetDist and returns the least sum of Euclidean distances. Next, Algorithm 2 ranks the least sum of Euclidean distances from each emotion and computes an exponential weighting for each emotion based on its ranking and the Euclidean distance. Lastly, Algorithm 3 estimates the AV values of the motion primitives with the weights found and the AV values in Table 1. Figure 7 shows keyframes from motion primitives that best reflect the emotions. Figure 8 shows the estimated emotion values of all motion primitives.

Algorithm 1 Calculate the least sum of Euclidean distances of points of interest of a motion primitive $M_{c,n}$ and the emotional static posture in SP[em]

GetLeastDiffEm($M_{c,n}, em$)

for KF = 1 to |SP[em]| **do**
  total ← 0
  for kf = 1 to numOfKeyframes($M_{c,n}$) **do**
    total ← total + GetDist($M_{c,n}$[kf], SP[em][KF])
  end for
  DIST[KF] ← total
end for
return $\min_{\text{KF}}$(DIST[KF])

Algorithm 2 Calculate the vector of weights based on the ranking of the Euclidean distances

GetWeightsBasedRank(distances)

for $i = 1$ to |distances| **do**
  flippedDistances[i] ← ($\sum_j$ distances[j]) − distances[i]
end for
sorted ← sortAscending(flippedDistances)
meanValue ← mean(flippedDistances)
for $i = 1$ to |flippedDistances| **do**
  weights[i] ← $e^k + \frac{\text{flippedDistances}[i]}{\text{meanValue}}$ where sorted[k] = flippedDistances[i]
end for
for $i = 1$ to |weights| **do**
  weights'[i] ← $\frac{\text{weights}[i]}{\sum_j \text{weights}[j]}$
end for
return weights'

5.1.2 The Markov Dancer Model

Suppose there is a dancer who dances with a piece of music. At each time point, the dancer strives both to reflect the emotion in the music and to achieve continuity of motions. We want to generate a schedule of motion primitives by mimicking this process. To be specific, this problem is modeled as a Markov chain, which is a generative stochastic motion model. A separate model (Figure 9) is used for for each category, e.g., Head. A Markov chain is used to select the motion primitives $M_{c,i}$ for each schedule S_c (S_{Head}, ..., S_{Legs}). We want to generate $M_{c,i}$ with the probability

Algorithm 3 Estimate AV value of $M_{c,n}$

GetActivationValence($M_{c,n}$)

for emID = 1 to 6 **do**
  emDiff[emID] ← GetLeastDiffEm($M_{c,n}$, emID)
end for
weights ← GetWeightsBasedRank(emDiff)
act ← 0
val ← 0
for emID = 1 to 6 **do**
  act ← act + weights[emID] ∗ em[emID].activation
  val ← val + weights[emID] ∗ em[emID].valence
end for
act ← bound(act, -1, 1)
val ← bound(val, -1, 1)

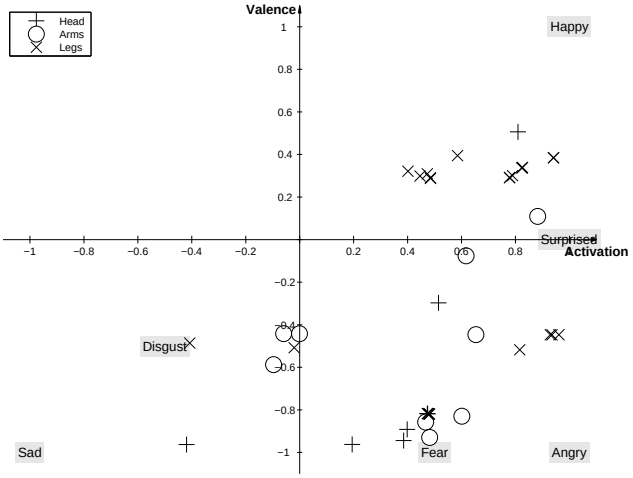


Figure 8: Labeled motion primitives (AV values)

$P(M_{c,i}|M_{c,i-1}, e)$, where e is the emotion detected at the end of $M_{c,i-1}$. As a special case, when $i = 1$, we select $M_{c,1}$ according to $P(M_{c,1}|e)$.

The motion primitive sequence generated by this model should (i) be continuous, (ii) reflect the musical emotion, and (iii) be interestingly non-deterministic. We set the probability function according to Equation 2.

$$P(M_{c,i}|M_{c,i-1}, e) = C \cdot E \cdot N \tag{2}$$

Here, we call C and E the *continuity factor* and *emotion factor*, respectively. They are based on the transition between different motion primitives and the emotion-motion primitive relationships. N is a constant normalizing factor.

Continuity factor: The continuity factor is designed to encourage continuity from each motion primitive to the next. Specifically, we want a quick and smooth interpolation from the last key frame of the current motion primitive to the first key frame of the next motion primitive. We denote the minimum required time interval computed from Algorithm 4 of this interpolation by $dist_M(M_{c,i+1}, M_{c,i})$ in Equation 3.

$$C = \frac{1}{\sqrt{2\pi\sigma_M^2}} \exp\left(-\frac{dist_M^2(M_{c,i+1}, M_{c,i})}{2\sigma_M^2}\right) \tag{3}$$

Here, σ_M^2 is a constant. The continuity factor is big when the minimum transition time is short.

Emotion factor: The emotion foctor is designed to se-

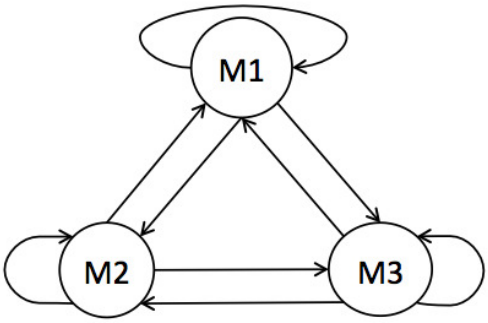


Figure 9: An example of the Markov dancer model shown with only 3 motion primitives for simplicity

lect motion primitives whose emotions are similar to the musical emotion. We denote the (a, v) coordinate of $M_{c,i}$ by $E(M_{c,i})$, and denote the Cartesian distance between $E(M_{c,i})$ and e on the AV plane by $dist_e(E(M_{c,i}), e)$ in Equation 4.

$$E = \frac{1}{\sqrt{2\pi\sigma_e^2}} \exp\left(-\frac{dist_e^2(E(M_{c,i}), e)}{2\sigma_e^2}\right) \quad (4)$$

Here, σ_e^2 is a constant. The emotion factor is big when the emotional difference is small. Again, e refers to the detected emotion at the end of $M_{c,i-1}$.

5.2 Determine Interpolation Times and Timing Parameters β in Schedule of Motion Primitives

After describing the process to select the sequence of motion primitives, we provide an algorithm to synchronize the schedule of motion primitives with the detected beat times, where each motion primitive in the schedule should end on a beat time. When a motion primitive ends, we begin interpolating to the first keyframe of the next motion primitive.

5.2.1 Calculate Time to Interpolate Between Motion Primitives

Algorithm 4 calculates the time needed to interpolate from the last keyframe, K_j, of the previous motion primitive $M_{c,m-1}$, to the first keyframe, K_l, of $M_{c,m}$, using the joint angles V_j of K_j and V_l of K_l. Although we can interpolate between two keyframes with maximum joint angular speeds given the joint angles, we want the robot to dance stably. As we do not implement the controller for the actuators of the robot to account for dynamics, we weight the minimum duration for the interpolation with a multiplier in Algorithm 4. We define λ as the maximum time multiplier, where $\lambda = 0.4$ ($e^{0.4} \approx 1.5$) so that the maximum time multiplier $\leq 1.5\gamma$. We define γ for each category (Table 2). E.g., we assign a higher γ of 3 for the legs and 1.5 for the head, so that the robot's legs move slower than the head and the robot is more stable at the bottom. We weighted the time multiplier more heavily when the $avgtime \approx maxtime$ which implies that all the joints move almost equally fast.

5.2.2 Calculate Timing Parameter β

The time required for each motion primitive includes the interpolation time between two primitives computed from Algorithm 4 and the times between the keyframes in the motion primitive. If only one beat-time interval is insufficient to execute the motion primitive, we add subsequent beat-time intervals until the total time offered is long enough

Algorithm 4 Calculate duration required to interpolate from K_j to K_l

GetDuration(K_j, K_l)
1: **for** JI = 1 to $|K_j|$ **do**
2: time[JI] $\leftarrow |V_{l,\text{JI}} - V_{j,\text{JI}}| \div \text{MaxAngularSpeed}[JI]$
3: **end for**
4: maxTime $\leftarrow$ max(time)
5: avgTime $\leftarrow$ average(time)
6: **if** maxTime = 0 **then**
7: **return** 0
8: **end if**
9: timeMultiplier $\leftarrow e^{\text{avgTime/maxTime}*\lambda} * \gamma$
10: **return** maxTime $*$ timeMultiplier

Table 2: γ values for joint categories

Category	Head	Arm	Leg
γ	1.5	2	3

for execution (Figure 10). To make each motion primitive end at a beat time, we stretch the duration by increasing the parameter, β, in each motion primitive to fill the time interval from its starting beat time to the next beat time. In practice, the schedule of motion primitives for each body part is planned independently and executed simultaneously.

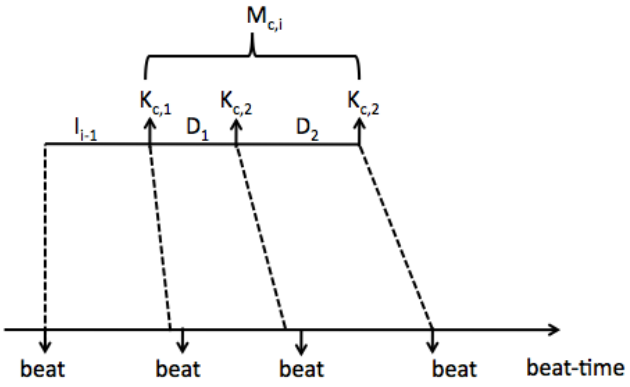


Figure 10: Synchronizing motion primitive with beat times

5.2.3 Emotion For Next Motion Primitive

Motion primitives are selected sequentially and stretched to fill a whole number of beat times. To choose the next motion primitive, we need the emotion at the end of the previous motion primitive. We simply estimate the emotion at each beat time by linearly interpolating the (a, v) values, which are computed at 15-second intervals.

6. EXECUTION

The schedule of motion primitives computed in Section 5.1 needs to be synchronized to the music based on beat times during execution. Since there are always latency and other differences between desired timing and real timing in robot motion, the starting and ending times of planned motion primitives will diverge from the plan and therefore become totally out of phase with respect to the music. To correct this drift, we use an adaptive real-time synchronizing algorithm (Algorithm 5), which is inspired by work on real-time

automatic music accompaniment [12]. Here, *start* and *end* are the two variables keeping track of ideal starting and ending times of each motion primitive $M_{c,m}$. Due to the execution error, the motion $M_{c,m-1}$ will not precisely end at its ideal ending time. Therefore, $M_{c,m}$ will start whenever $M_{c,m-1}$ ends and we recalculate the *duration* from the actual real time returned by the function, getTime. The duration is calculated so that $M_{c,m}$ will end at the ideal end time. Again, $M_{c,m}$ will not actually finish at exactly this ideal ending time. Therefore, $M_{c,m+1}$ will again calculate a *duration* and move on. This algorithm adjusts the execution duration of every motion primitive by updating the parameter, β, in the motion primitive using the function updateBeta to avoid the accumulated timing errors. updateBeta adjusts the total time that it originally takes to execute the motion primitive to be the same as the updated *duration* by changing β of the motion primitive.

Algorithm 5 Adaptive Real-time Synchronizer

Synchronizer(S_c)
1: *start* ← getTime()
2: **for all** $M_{c,m}$ in S_c **do**
3: *end* ← *start* + $M_{c,m}$.duration
4: *duration* ← *end* − getTime()
5: updateBeta($M_{c,m}$, *duration*)
6: execute($M_{c,m}$)
7: *start* ← *end*
8: **end for**

7. RESULTS

Table 3 is a contrast experiment to show how the continuity and emotion factors affect the plan for a Pleased piece of music and right arm motion primitives as an example. The first column is the average minimum duration for the interpolation from one motion primitive to the next one. Smaller numbers indicate greater continuity. The second column is the average Euclidean distance between the emotion of the motion primitives and the emotion of the music on the AV plane. Smaller numbers indicate greater correspondence between the dance emotion and the music emotion. The first row is the experimental trial, which takes both the continuity and emotion factors into account, while the second and the third row are control trials, which eliminate the continuity factor and emotion factor, respectively. The fourth row is also a control trial, which eliminates both factors and generates a random dancing plan. The results show that both emotion and continuity factors are beneficial.

Table 3: A contrast experiment to show how continuity and emotion affect dancing plan

Trial	Behavior distance	Emotion distance
E and C	0.55	0.61
E	0.94	0.49
C	0.4	0.63
R	0.79	0.68

Figure 11 shows a planned schedule of motion primitives for the *RArm* for a short snippet of Peaceful music whereas Figure 12 shows a planned schedule of motion primitives the *RArm* for Angry music. Figure 11 and Figure 12 plot the music signals and beat times and we show that the planned schedule of motion primitives corresponds to the beats.

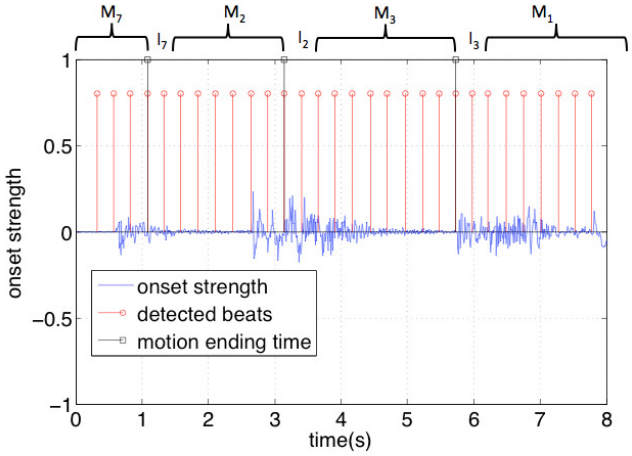


Figure 11: RArm motion primitives schedule for Peaceful music

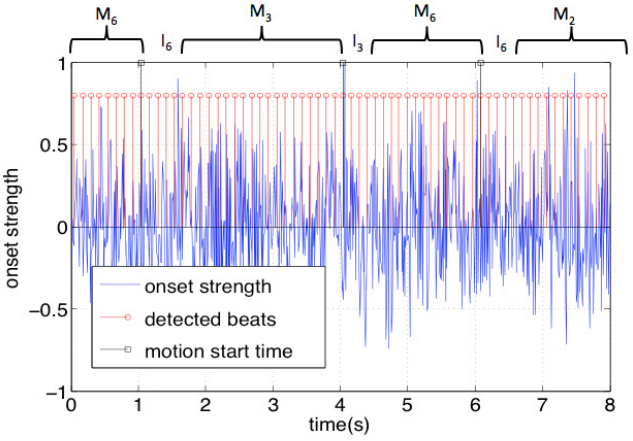


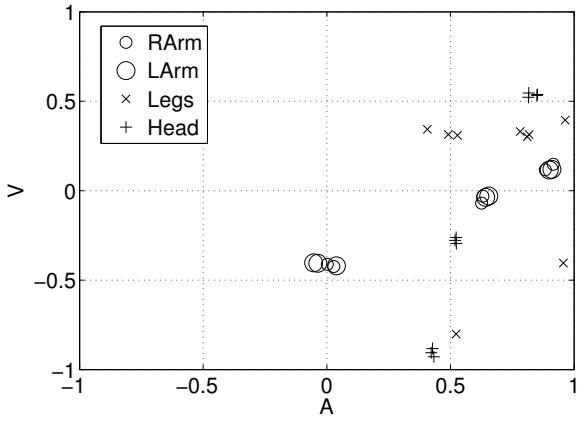
Figure 12: RArm motion primitives schedule for Angry music

We also show that different motion primitives are selected for Pleased music and Angry music in Figure 13. Figure 14 shows how the real-time synchronizer reduces timing errors. The dotted line shows executed time using real-time synchronizer and is a good match to the ideal time. The black line shows the executed time without a real-time synchronizer, which is totally out of phase after a minute.

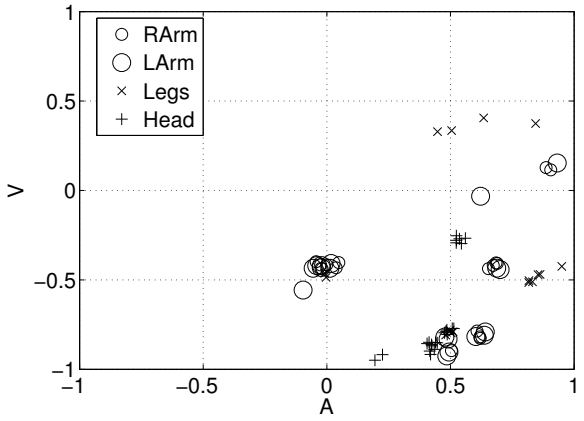
Figure 15 shows snapshots of the NAO humanoid robot dancing with a piece of Angry music. Most of these snapshots show the NAO robot leaning forward, the head bent forward, and putting the arms at the side of the body. These postures are similar to the Angry static postures collected.

8. CONCLUSIONS

We show that we can automate robot dancing by forming schedules of motion primitives that are driven by the emotions and the beats of any music on a NAO humanoid robot. The algorithms are general and can be used on any robot. From emotion labels given for static postures, we can estimate the activation-valence space locations of the motion primitives and select the appropriate motion primitives for emotions detected in music. We also show that we can monitor the execution of the schedule of motion primitives and compensate for any timing errors found, ensuring synchro-



(a)



(b)

Figure 13: Activation-Valence coordinates of motion primitives chosen for (a) Pleased music, and (b) Angry music. Motion primitives visited multiple times are drawn with slight offsets to convey their number.

nization between robot dance motions and the music.

Acknowledgments

This work is supported by a generous gift awarded to the School of Computer Science, Carnegie Mellon University. We wish to thank Byeong-jun Han for the comments on music emotion recognition. We also wish to thank Somchaya Liemhetcharat for his help on data collection and feedback on the algorithms.

9. REFERENCES

[1] S. Bock and M. Schedl. Enhanced beat tracking with context-aware neural networks. In *Proc. Int. Conf. Digital Audio Effects*, 2011.

[2] C. Breazeal. Emotion and sociable humanoid robots. *Int. J. of Human-Computer Studies*, 59:119–155, 2003.

[3] P. Ekman. Are there basic emotions? *Psychological Review*, 99(3):550–553, 1992.

[4] D. Ellis. Beat tracking with dynamic programming. *MIREX Audio Beat Tracking Contest sys. desc.*, 2006.

[5] A. J. Eronen and A. P. Klapuri. Music tempo estimation with k-nn regression. *Trans. Audio, Speech and Lang. Proc.*, 18(1):50–57, 2010.

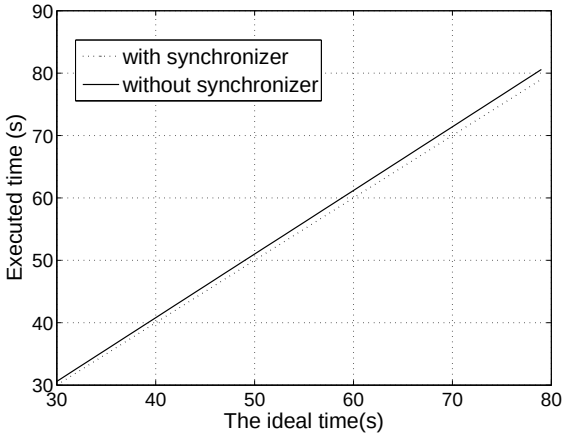


Figure 14: Executed time difference: with a real-time synchronizer (RTS) vs. no RTS

Figure 15: Video keyframes of NAO humanoid robot dancing with Angry music

[6] M. Goto. A study of real-time beat tracking for musical audio signals. *Ph.D. thesis*, 1998.

[7] B. Han, S. Rho, R. Dannenberg, and E. Hwang. SMERS: Music emotion recognition using support vector regression. In *ISMIR'09*, pages 651–656, 2009.

[8] G. Kim, Y. Wang, and H. Seo. Motion control of a dancing character with music. In *IEEE/ACIS Int. Conf. Comp. Info. Science*, pages 930–936, 2007.

[9] T.-h. Kim, S. Park, and S. S. Y. Rhythmic-motion synthesis based on motion-beat analysis. *ACM Transactions on Graphics*, 2003.

[10] R. Kirby, R. Simmons, and J. Forlizzi. Modeling affect in socially interactive robots. In *Proc. Int. Symp. Robot Human Interact. Comm.*, pages 558–563, 2006.

[11] H. Lee and I. Lee. Automatic synchronization of background music and motion. *Computer Graphics Forum*, 24:353–362, 2005.

[12] D. Liang, G. Xia, and R. Dannenberg. A framework for coordination and synchronization of media. In *Proc. Int. Conf. New Interfaces Musical Expr.*, 2011.

[13] S. Nakaoka, S. Kajita, and K. Yokoi. Intuitive and flexible user interface for creating whole body motions of biped humanoid robots. In *IEEE Int. Conf. Intelligent Robots and Systems*, pages 1675–1682, 2010.

[14] J. Oliveira, L. Naveda, F. Gouyon, M. Leman, and L. Reis. Synthesis of variable dancing styles based on a compact spatiotemporal representation of dance. In *IEEE Int. Conf. Intelligent Robots and Systems*, 2010.

[15] T. Shiratori, A. Nakazawa, and K. Ikeuchi. Dancing-to-music character animation. *Computer Graphics Forum*, 25:449–458, 2006.

[16] R. E. Thayer. *The Biopsychology of Mood and Arousal*. Oxford University Press, New York, 1989.